DirectNET PLC Access

Local application

Thu, Feb 1, 1996

The vacuum controls interface for the PET project uses a Programmable Logic Controller to do the interlocks handling and vacuum-specific logic that is required. The IRM interfaces to the PLC via an RS-232 serial port. The basic approach is to routinely collect analog and digital data from the PLC, then map it into the IRM's analog and digital channels. Control actions are also output as necessary. All of this logic is handled by a local application.

The serial I/O input supported by the system system software passes through the Serial Input Queue (SERIQ) table. By monitoring the contents of the SERIQ, all received characters of serial input, except the linefeed (0A) and null (00) characters can be seen. This is enough to catch the data coming from the PLC.

The DirectNET protocol can transmit data in hex (binary) or in Ascii. Although using Ascii requires twice the time for the data transfers, it helps to unambiguously detect control characters that are part of the protocol. This implementation of DirectNET support will use Ascii for that reason. The following control codes are used by the DirectNET protocol:

ENQ	05	ETB	17
ACK	06	STX	02
NAK	15	ETX	03
SOH	01	EOT	04

DirectNET Overview

One data transaction requires a series of I/O communications between the host computer and the slave PLC. To begin any transaction, the master sends an inquiry 3-byte sequence of "N", "address", ENQ, where address = $20 + the PLC slave address. The slave responds with the same sequence, with the ENQ byte replaced by an ACK.

The master then sends a header that defines the operation. Its format is SOH, header, ETB, LRC. The LRC stands for a one byte Longitudinal Redundancy Check that is the exclusive OR of all the Ascii bytes within the header. The header itself consists of the one-byte (two Ascii characters) slave address, read (30) or write (38) character, data type character, two-byte starting address, one-byte #complete (256-character) blocks, one-byte number of bytes in last block, and one master ID byte (0 or 1). The slave responds with an ACK character.

For a read request, the slave continues by sending the data message in the format STX, data block, ETX, LRC. If the #characters in the data block is larger than 256, so that the #complete data blocks is nonzero, then more than one data block is sent, with each complete data block using a ETB character in place of the ETX. Each data word within a data block is in byte order, least byte first. After each complete data block, or the last incomplete data block (of length 0–255 characters) is received by the master, the master returns an ACK. The slave then returns an EOT. The master finally sends an EOT. This final EOT clears the slave for future detection of an inquiry sequence.

For a write request, the transaction sequence is the same, but the data is transferred from the master and ACK'd by the slave. After the last data block is ACK'd by the slave, then the master sends an EOT to end the entire transaction.

Timeouts are imposed on the successive communications of a transaction. If a slave times out awaiting a response from a master, it will be necessary for the master to send an EOT to clear the slave to accept a new inquiry.

The details of this communication protocol are found in the manual.

IRM serial support

The usual serial port support in an IRM is organized around lines of input separated by a CR character. Nulls (00) and LF characters (0A) are removed from the input stream. If more than 128 characters are received without a CR, then one is inserted into the serial stream. By operating the DirectNET communications in Ascii mode, this should not cause a problem, as LF and nulls and CR aren't used. The slave sends a CR in Ascii mode. But a data block could be longer than 128 characters, so waiting for one would not be advisable. Therefore, as a first step, we will simply monitor (at 10Hz) what is found inside the SERIQ and in this way be able to see all characters in the stream as soon as they come in and are deposited into the SERIQ by the serial receive interrupt code.

Serial output support is usually organized as lines, with trailing blanks removed and CR and LF inserted. This is the usual way, but there is a separate listype that permits serial output without such editing. We shall use the latter listype for DirectNET output, in case the CR, LF would cause a problem for the slave PLC. The serial baud rate for use with the DirectNET interface is 19200 baud.

Data acquisition approach

The DNET local application program is used to collect the data routinely by sending a read transaction. The response data consists of two parts, the first for analog and the second for digital data. The response data is then mapped into the IRM's local analog channels and digital bytes. Between data acquisition transactions, DNET also monitors a message queue for setting commends, either to an analog word or a digital word. When a message is detected, a write transaction is made in place of the next data acquisition transaction. This approach means that all the support afforded analog channels and binary bits in the IRM system can be preserved. The acquisition may be slow, but this is not thought to be a problem for a vacuum system controls interface. With this approach, an update rate of 1Hz or better, and a control action delay of less than one second, should be achievable.

In order to prevent other uses of the serial port for output, we may place a flag bit in the PRNTQ header that prevents such output. A simple way to do this may be to allow only the raw listype to work for serial output. Usual serial port output uses the normal output logic that edits out terminal blanks and adds CR,LF.

Message queue support

A change in the system code supports use of a PLCQ message queue. When a setting is made to a PLC-type device, a message about the setting is placed into the message queue. (If it has not been created, it will first be created.) In this way, there is a place for the settings that result for the Restore action following a system reset to reside, until the time that the DNET local application is initialized and the first data acquisition transaction completed. As DNET is initialized, it attaches to the PLCQ message queue so it can check for any waiting messages.

Parameters

Local application DNET parameters, using example test values, are as follows:

```
ENABLE  B 00D4    Bit# enables local application
SLAVE0001 Slave address of PLC interface
DATATYPE  0001    Data type# used for data pool acquisition
REFADDR   1001    Base reference addr for analog, digital data pool
NACHANS   0010    #chans of analog data
NDWORDS   0008    #words of digital data following analog data
MAPCHAN C 0180    Base analog Chan# for mapping to local IRM space
MAPBIT  B 0180    Base binary Bit# for mapping to local IRM space
          0000 (spare)
          0000 (spare)
```

The above set of parameter values supports 16 analog channels and 8 words (128 bits) of digital data.

Digital control scheme

Each BADDR entry is normally a memory address that should be written for the associated status byte. But 1553 and SRM communications required specially-coded 4-byte BADDR entries that are signaled by the use of hi byte values 80 and 81, respectively. For the PLC support, we use a hi byte value of 82. When the usual binary data scan occurs, via the "0405" entry in the data access table, such entries are skipped. When a digital control setting is made, the data type and reference address are found in the lower three bytes of the BADDR entry. To perform the setting, the information must be passed to the DNET local application via the message queue scheme described above.

DNET collects the data pool from the PLC every 4 cycles. For support by a local application that is invoked at 10 Hz, this is the easiest approach. During the first cycle, the enquiry message is sent. On the second cycle, the 3-byte response to the enquiry is received, and the request header is sent. On the third cycle, the acknowledgment to the request header is received, followed by the data that was requested, and the ACK is sent. (If there is too much data, given the bandwidth available, then an additional cycle or more would be required.) On the fourth cycle, the EOT is received, and the EOT is sent to the PLC to clear it for receipt of the next enquiry. The message queue is checked for any settings to be performed. If one is found, then another four cycles is spent doing that write transaction. The required data type byte, reference address word, and data word are taken from the message queue entry, which was filled by the setting support in the system code using the contents of the BADDR entry. Upon completion of the write transaction, a new read transaction is performed that updates the data pool. As a result, the data pool is updated every 0.4 seconds, but when a setting must be performed, 0.4 seconds is taken to perform it. The maximum time between updates of the data pool is therefore 0.8 seconds. The maximum time to perform a setting, assuming none is already queued, is also 0.8 seconds. If a faster update rate is needed, a means of invoking the local application in response to serial activity will be required. At first, omitting such support is easier.

Bit-based, byte-based, and word-based digital control are supported. In any case, however, a word-wide setting is actually performed. Bit-based toggle, set hi, and set lo digital control types are supported. Bit-based pulse types are not supported for this hardware; the PLC's cpu logic can be used to do it.

Analog control

Analog control is specified by a new analog control type# $19. The second byte gives the data type, and the last two bytes give the reference addresss to be used to effect the setting. It may be in the memory region that is part of the data pool, in which case the PLC's cpu will have to perform the setting to the real I/O module; or it may be in the I/O module itself. Upon successfully queuing the setting message, the setting word of the ADATA entry for that channel is updated, even though completely successful completion of the setting is not assured. Because knob control could queue settings faster than they can be delivered at 0.8 sec, the local application checks for successive entries in the queue referencing the same target address (data type and reference address), and coalesces them as much as possible, delivering only the final setting it finds waiting in the queue.